# 09-Mutable vs Immutable string

## Mutable vs. Immutable Strings in Java

### Memory Allocation for Strings

In Java, memory management is crucial, especially when dealing with strings. Understanding how strings are allocated and managed in memory can help in optimizing performance and avoiding unnecessary overhead.

**Example:**

```
class Example {
   public static void main(String[] args) {
      String name = "navin";
      System.out.println(name + " reddy");

      String s1 = "navin";
      String s2 = "navin";
      System.out.println(s1 == s2); // Checking if both are referring to the same object or not
   }
}
```

**Output:**

```
navin reddy
true
```

## Explanation

At first glance, you might assume that s1 and s2 refer to different objects. However, both s1 and s2 point to the same object in memory. This is due to how Java handles string literals.

## Memory Allocation in JVM

Within the JVM, memory is divided into two main regions: **Heap Memory** and **Stack Memory**. When it comes to strings, there is a special area within the heap known as the **String Constant Pool (SCP)**.

- **String Constant Pool**: This pool is responsible for managing string literals. Whenever a string literal is created, the JVM checks the SCP. If the string already exists in the pool, the same object is reused, and no new memory is allocated. If the string is not found in the SCP, a new memory allocation occurs.

## What is the String Pool?

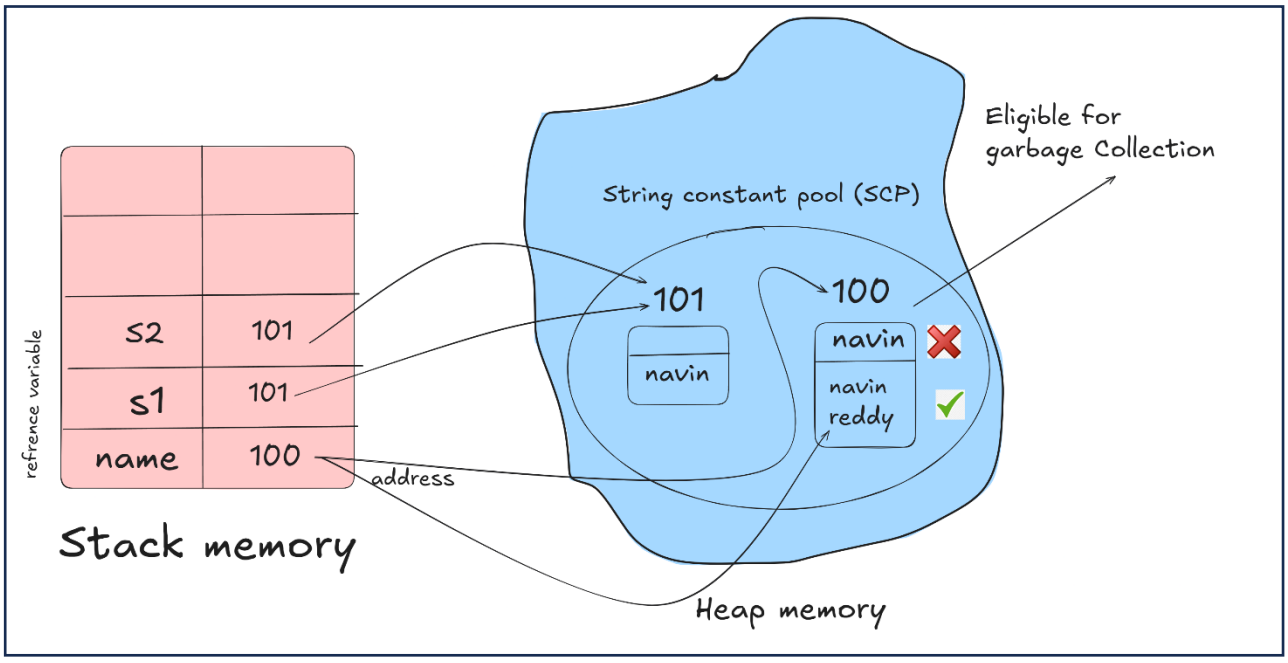
The String Pool (or String Intern Pool) is a special storage area in the Java heap where string literals are stored. It's an optimization mechanism to reduce memory usage and improve performance. By default, the String Pool is empty and is managed by the Java String class.

- **Process**:
    - When a string literal is created, the JVM checks the String Pool.
    - If the literal exists, the JVM returns a reference to the existing object.
    - If the literal doesn't exist, a new string object is created and added to the pool.

## Example Breakdown

In the example provided:

- The string name is initially set to "navin". When "navin" is appended with " reddy", a new memory allocation occurs because strings in Java are immutable.
- The original "navin" object becomes eligible for garbage collection since it's no longer referenced.




This demonstrates **string immutability**—once a string object is created, its value cannot be changed. Instead, any modification creates a new string object.

## Mutable Strings

For scenarios where string modification is required without creating new objects, Java provides:

- **StringBuilder**

- **StringBuffer**

Both of these classes allow for mutable strings, meaning you can modify the string without creating new objects in memory.

**Example of Mutable Strings**

```
StringBuilder sb = new StringBuilder("navin");

sb.append(" reddy");

System.out.println(sb.toString());
```

**Output:**

```
navin reddy
```

In this case, StringBuilder allows the string to be modified without creating a new object.